



University of
Pittsburgh

Algorithms and Data Structures 2

CS 1501



Spring 2023

Sherif Khattab

ksm73@pitt.edu

Announcements

- Upcoming Deadlines
 - Homework 2: this Friday @ 11:59 pm
 - Lab 1: tomorrow @ 11:59 pm
 - Assignment 1: Friday 2/17 @ 11:59 pm
- Lecture recordings are available on Canvas under Panopto Video
- Please use the “Request Regrade” feature on GradeScope if you have any issues with your homework grades
- TAs student support hours available on the syllabus page

Previous lecture

- Binary Search Tree

This Lecture

- Binary Search Tree
 - How to search, add, and delete
- Runtime of BST operations

Reflect on the steps that you followed

- How did you add 4 to the tree?
- What steps did you follow?

10, 8, 17, 7, 5, 20, 15, 16, 4

BST: How to add?

- How to add a data item *entry* into a BST rooted at *root*?
- What if `root.data.compareTo(entry) == 0`?
- What if `root.data.compareTo(entry) < 0`?
 - Move left or right?
 - What if no child?
 - What if there is a child?
- What if `root.data.compareTo(entry) > 0`?
 - Move left or right?
 - What if no child?
 - What if there is a child?
- What if I tell you that you have a friend who can add into a BST.
 - How can you use the help of that friend?

Let's see the code for adding into a BST

- Available online at:
 - <https://cs1501-2231.github.io/slides-handouts/CodeHandouts/TreeADT/Slides>
 - The slides are under the CodeHandouts/TreeADT/slides folder in the handout repository
 - <https://github.com/cs1501-2231/slides-handouts>

Let's build a Binary Search Tree

- Work in groups of 2-3 students
- Add the following integers to a Binary Search Tree in **the following order**:

4, 5, 7, 8, 10, 8, 15, 16, 17, 20

Reflect on the steps that you followed

- How many comparisons did you have to make to add 20?

4, 5, 7, 8, 10, 8, 15, 16, 17, 20

Run-time of BST operations

- For add, # comparisons = d , where d is the depth of the new node
- For search miss, # comparisons = d , where d is the depth of the node if it were in the tree
- For search hit, # comparisons = $1+d$, where d is the depth of the found node
- On average, node depth in a BST is $O(\log n)$
 - n is the number of data items
 - Proof in Proposition C in Section 3.2 of Sedgewick Textbook
- In the worst case, node depth in a BST is $O(n)$

Let's switch to delete!

- Work in groups of 2-3 students
- In the Binary Search Tree that you built out of **the following order**:

10, 8, 17, 7, 5, 20, 15, 16, 4

- How would you delete 4?
- How would you delete 5?
- How would you delete 10?

BST: How to delete?

- Three cases
 - Case 1: node to be deleted is a leaf
 - Easiest case!
 - Pass back null to the node's parent
 - Case 2: node to be deleted has one child
 - Pass back the child to the node's parent to adopt instead of the node

BST: How to delete?

- Three cases
 - Case 3: node to be deleted has two children
 - Difficult to delete the node!
 - It has two children, so parent cannot adopt both
 - Let's try to replace the data in the node
 - We still want to maintain the search tree property
 - What are our options?
 - Replace node's data by its successor
 - largest item in left subtree
 - or by its predecessor
 - smallest item in right subtree
 - Delete the node that we selected in the previous step
 - Has to have at most one child
 - Why?

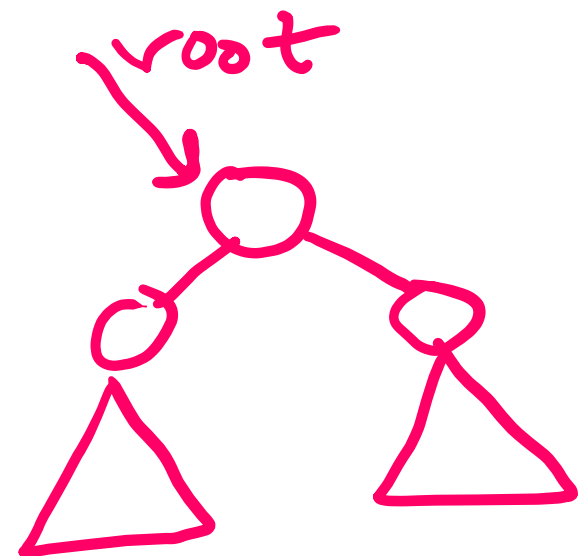
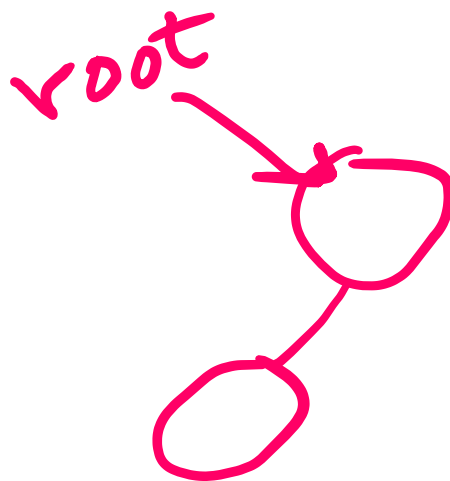
Let's see the code for deleting from a BST

- Available online at:
 - <https://cs1501-2231.github.io/slides-handouts/CodeHandouts/TreeADT/Slides>
 - The slides are under the CodeHandouts/TreeADT/slides folder in the handout repository
 - <https://github.com/cs1501-2231/slides-handouts>

BST: delete operation

- Deleting an item requires first to find the node with that item in the tree
- Let's assume that we have already found that node
- The method below returns a reference to the root of the tree after removing its root

```
private BinaryNode<T> removeFromRoot(BinaryNode<T> root){
```



Delete Case 1: tree has only one node

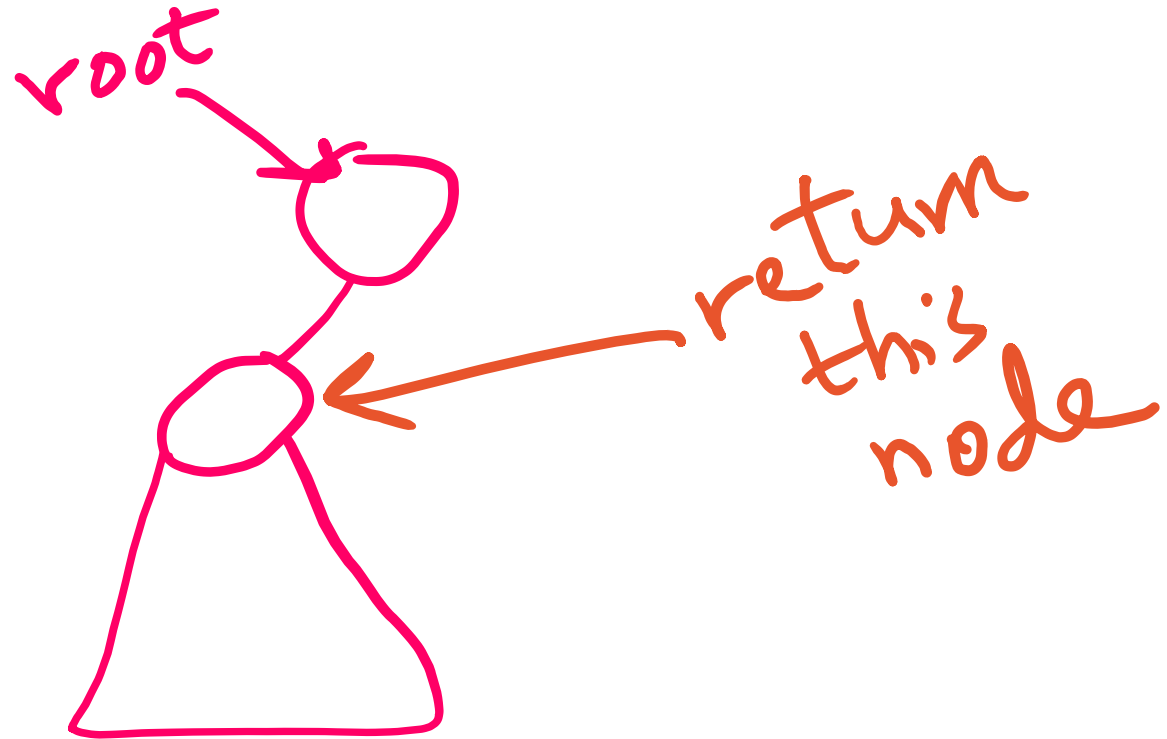
```
private BinaryNode<T> removeFromRoot(BinaryNode<T> root){
```



Return null

Delete Case 1: root has one child (left or right)

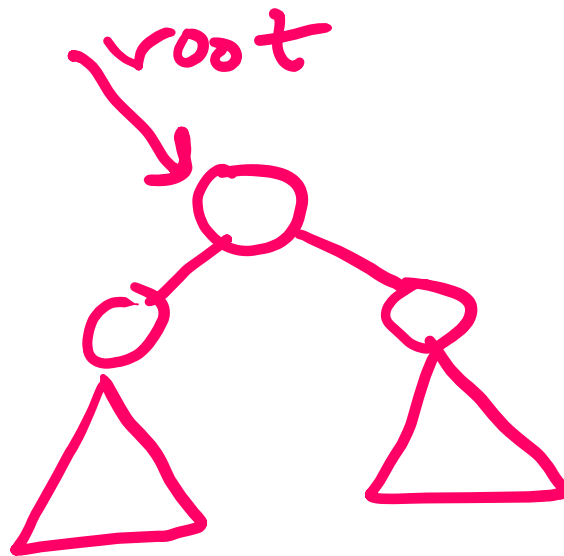
```
private BinaryNode<T> removeFromRoot(BinaryNode<T> root){
```



Return the root of the subtree rooted at the child

Delete Case 1: root has two children

```
private BinaryNode<T> removeFromRoot(BinaryNode<T> root){
```



- replace root's data by the data of the largest item of its left subtree (why?)
- remove the largest item from the left subtree
- return root